Schemas for safe and efficient XML processing

Dario Colazzo

Université Paris Sud - INRIA

Ecole Thématique BDA 2010 - Les Houches
Plan

• XML & XML schema

• Correctness and result analysis

• Schema based projection

• Constraint based approach for efficient subtyping and validation
Plan

- XML & XML schema
- Correctness and result analysis
- Schema based projection
- Constraint based approach for efficient subtyping and validation.

Informal, examples, and main ideas
~ 15 minutes
Plan

- XML & XML schema
- Correctness and result analysis
- Schema based projection
- Constraint based approach for efficient subtyping and validation

Informal, examples, and main ideas
~ 15 minutes

More formal, examples + some defs
~ 30 minutes
What are XML schemas useful for?

- To define structural constraints over documents: this is useful in many contexts.
- How: mainly by means of regular expressions.
- Main schema languages: DTDs, XML Schema, Relax-NG.
- For all of them, methods for automatic validation exist.
- For XML queries over XML valid documents we can
  - automatically check that the query correctly manipulate the input
  - automatically infer a schema for data produced by the query
XML query type-checking
Query correctness

The quite famous biblio DTD

```xml
<!ELEMENT bib (book* )>
<!ELEMENT book (title, (author+ | editor+ ), publisher, price )>
<!ATTLIST book year CDATA #REQUIRED>
<!ELEMENT author (last, first )>
<!ELEMENT editor (last, first, affiliation )>
<!ELEMENT title (#PCDATA )>

Query:

```xml
<res>
  for x in doc//(author | editor)
  return <nom>x/last</nom>
</res>
```
Query correctness

The quite famous biblio DTD

```xml
<!ELEMENT bib (book* )>
<!ELEMENT book (title, (author+ | editor+ ), publisher, price )>
<!ATTLIST book year CDATA #REQUIRED >
<!ELEMENT author (last, first )>
<!ELEMENT editor (last, first, affiliation )>
<!ELEMENT title (#PCDATA )>
......
```

**Query:**

```xml
<res>
 for x in doc//(author | editor)
 return <nom>x/last</nom>
</res>
```

Is it correct? Yes, intuitively

Not according to the traditional $\forall$-correctness

Problems with an $\exists$-notion
Query correctness

The quite famous biblio DTD

```xml
<!ELEMENT bib (book* )>
<!ELEMENT book (title, (author+ | editor+), publisher, price )>
<!ATTLIST book year CDATA #REQUIRED >
<!ELEMENT author (last, first )>
<!ELEMENT editor (last, first, affiliation )>
<!ELEMENT title (#PCDATA )>
......
......
```

**Query:**

```
<res>
  for x in doc//(author | editor)
  return <nom>x/last</nom>
</res>
```

Is it correct? Yes, intuitively

Not according to the traditional $\forall$-correctness

Problems with an $\exists$-notion

We need $\forall$ quantification on sub-queries

and $\exists$ quantification on instances
Query correctness

The quite famous biblio DTD

<!ELEMENT bib (book* )>

<!ELEMENT book (title, (author+ | editor+ ), publisher, price )>

<!ATTLIST book year CDATA #REQUIRED >

<!ELEMENT author (last, first )>

<!ELEMENT editor (last, first, affiliation )>

<!ELEMENT title (#PCDATA )>

......

Query:

<res>
for x in doc//(author | editor)
return <nom>x/last</nom>
</res>

Is it correct? Yes, intuitively

Not according to the traditional \( \forall \)-correctness

Problems with an \( \exists \)-notion

We need \( \forall \) quantification on sub-queries \( \forall \exists \) correctness

and \( \exists \) quantification on instances [PhD Thesis, ICFP04]
Query correctness

The quite famous biblio DTD

```xml
<!ELEMENT bib  (book* )>
<!ELEMENT book  (title, (author+ | editor+ ), publisher, price )>
<!ATTLIST book  year CDATA  #REQUIRED >
<!ELEMENT author  (last, first )>
<!ELEMENT editor  (last, first, affiliation )>
<!ELEMENT title  (#PCDATA )>
......
```

Query:

```xml
<res>
  for x in doc//{(author | editor)}
  return <nom>x/last</nom>
</res>
```

Is it correct? Yes, intuitevely

Not according to the traditional ∀-correctness

Problems with an ∃-notion

We need ∀ quantification on sub-queries and ∃ quantification on instances

[PhD Thesis, ICFP04]
Query correctness

The quite famous biblio DTD

```xml
<!ELEMENT bib (book* )>
<!ELEMENT book (title, (author+ | editor+), publisher, price )>
<!ATTLIST book year CDATA #REQUIRED >
<!ELEMENT author (last, first )>
<!ELEMENT editor (last, first, affiliation )>
<!ELEMENT title (#PCDATA )>
......
```

**Query:**

```xml
<res>
for x in doc//(author | editor)
return <nom>x/last</nom>
</res>
```

Is it correct? Yes, intuitively

Not according to the traditional \( \forall \)-correctness

Problems with an \( \exists \)-notion

We need \( \forall \) quantification on sub-queries \( \forall \exists \) correctness

and \( \exists \) quantification on instances

[PhD Thesis, ICFP04]
Main tools for ∀∃ correctness

• A type system allowing to infer types of query paths:
  • `doc//(author | editor) : (author | author)+`
  • `doc//(author | editor)/second : (second)+`

• As a consequence, the type system allows to find types of elements never needed by the query (all XPath axes can be handled)

• This has been used for type-based projection: first types of needed nodes are inferred, and then this information is used to prune the input D in order to obtain a much smaller document D′ such that

\[ Q(D) = Q(D') \]
Type based projection
Example

<!ELEMENT bib (book* )>
<!ELEMENT book (title, (author+ | editor+ ), publisher, price )>
<!ATTLIST book year CDATA #REQUIRED >
<!ELEMENT author (last, first )>
<!ELEMENT editor (last, first, affiliation )>
<!ELEMENT title (#PCDATA )>
......
......

Query:

<res>
for x in doc//(author | editor)
return <nom>x/last</nom>
</res>
Example

```xml
<!ELEMENT bib (book*) >

<!ELEMENT book (title, (author+ | editor+ ), publisher, price ) >

<!ATTLIST book year CDATA #REQUIRED >

<!ELEMENT author (last, first ) >

<!ELEMENT editor (last, first, affiliation ) >

<!ELEMENT title (#PCDATA )>

......

......

Type projector \( \tau = (\text{bib, book, author, editor, last}) \)

Used at loading time: only \( \tau \) elements are kept, the other ones are not loaded
Much less memory consumption: we can query quite big documents!

<bib>
  <book year="1994">
    <title>TCP/IP Illustrated</title>
    <author><last>Stevens</last><first>W.</first></author>
    <publisher>Addison-Wesley</publisher>
    <price>65.95</price>
  </book>

  <book year="1992">
    <title>Advanced Programming in the Unix environment</title>
    <author><last>Stevens</last><first>W.</first></author>
    <publisher>Addison-Wesley</publisher>
    <price>65.95</price>
  </book>

  <book year="2000">
    <title>Data on the Web</title>
    <author><last>Abiteboul</last><first>Serge</first></author>
    <author><last>Buneman</last><first>Peter</first></author>
    <author><last>Suciu</last><first>Dan</first></author>
    <publisher>Morgan Kaufmann Publishers</publisher>
    <price>39.95</price>
  </book>

  <book year="1999">
    <title>The Economics of Technology and Content for Digital TV</title>
    <editor>
      <last>Gerbarg</last><first>Darcy</first>
      <affiliation>CITI</affiliation>
    </editor>
    <publisher>Kluwer Academic Publishers</publisher>
    <price>129.95</price>
  </book>
</bib>
Test results: space [VLDB06]

Memory (in MB)

Query

XMark QMi
XPathMark QPj
Test results: time [VLDB06]

XMark QMi
XPathMark QPj
What about updates?

- Type-based projection still ensures optimizations
- Amine Baazizi will give you more details
- Marina Sahakyan can answer questions about efficient implementation of the technique
Let’s go back to type inference

• Very important problem, crucial for **result analysis**

• Given Q over a schema S, does Q produce values of another expected schema S’

\[ Q : S \rightarrow S' \ ? \]

• Method:
  • automatic inference of a schema $S_{out}$ for Q result values
  • automatic checking of inclusion $S_{out} \subseteq S'$

• Problem: schema inclusion has high complexity.

• We found out that for a wide class of schemas it can be efficiently checked. Next subject.
Constraints based subtype checking and validation
REs and XML types

- REs define element content models in XML schemas

  DTD:  
  
  <!ELEMENT book (title, (author | editor)*, price?)>
REs and XML types

- REs define element content models in XML schemas

  DTD:  
  
  ```
  <!ELEMENT book (title, (author | editor)*, price?)>
  ```
REs and XML types

- REs define element content models in XML schemas
  
  DTD:  `<!ELEMENT book (title, (author | editor)*, price?)>`

- Our syntax

  \[ T ::= \varepsilon \mid a \mid T + T \mid T \cdot T \mid T^* \]
REs and XML types

- REs define element content models in XML schemas
  
  DTD :  `<!ELEMENT book (title, (author | editor)*, price?)>`

- Our syntax

  \[ T ::= \varepsilon \mid a \mid T + T \mid T\cdot T \mid T^* \]

  \[ \text{title} \cdot (\text{author} + \text{editor})^* \cdot (\text{price} + \varepsilon) \]
Interleaving and counting

\[ T ::= \varepsilon \mid a \mid T + T \mid T \cdot T \mid T^* \mid T \& T \mid T[n..m] \]
Interleaving and counting

\[ T ::= \varepsilon \mid a \mid T + T \mid T \cdot T \mid T^* \mid T&T \mid T[n..m] \]

- \( m \in \mathbb{N}\cup\{\ast\} \)
Interleaving and counting

T ::= ε | a | T + T | T·T | T* | T&T | T[n..m]

• m ∈ N∪{*}

• a = a[1..1]  a? = a+ε  a* = a[1..*]
Interleaving and counting

\[ T ::= \varepsilon \mid a \mid T + T \mid T \cdot T \mid T^* \mid T^n \mid T^m \]

- \( m \in \mathbb{N} \cup \{\ast\} \)
- \( a = a[1..1] \quad a? = a+\varepsilon \quad a^* = a[1..\ast] \)
- \( L(b[1..4]) = \{b, bb, bbb, bbbb\} \)
- \( L(a \& b) = \{ab, ba\} \)
Interleaving and counting

\[ T ::= \varepsilon \mid a \mid T + T \mid T \cdot T \mid T^* \mid T\&T \mid T[n..m] \]

- \( m \in \mathbb{N} \cup \{\ast\} \)
- \( a = a[1..1] \quad a?=a+\varepsilon \quad a^*=a[1..\ast] \)
- \( L(b[1..4])=\{b, bb, bbb, bbbb\} \)
- \( L(a\&b)=\{ab, ba\} \)
Interleaving and counting

\[
T ::= \epsilon \mid a \mid T + T \mid T \cdot T \mid T^* \mid T \& T \mid T[n..m]
\]

- \( m \in \mathbb{N} \cup \{\ast\} \)
- \( a = a[1..] \quad a? = a+\epsilon \quad a* = a[1..\ast] \)
- \( L(b[1..4]) = \{b, bb, bbb, bbbb\} \)
- \( L(a\&b) = \{ab, ba\} \)
- \( L((a \cdot b) \& c) = \{abc, cab, acb\} \quad cba \notin L((a \cdot b) \& c) \)
Interleaving and counting

\[ T ::= \varepsilon \mid a \mid T + T \mid T \cdot T \mid T^* \mid T\&T \mid T[n..m] \]

- \( m \in \mathbb{N}\cup\{\ast\} \)
- \( a = a[1..1] \quad a? = a+\varepsilon \quad a^* = a[1..\ast] \)
- \( L(b[1..4]) = \{b, bb, bbb, bbbb\} \)
- \( L(a\&b) = \{ab, ba\} \)
- \( L((a \cdot b) \& c) = \{abc, cab, acb\} \quad cba \notin L((a \cdot b) \& c) \)
Interleaving

- Interleaving is used in XML type languages
- RELAX-NG `<interleave> … </interleave>`
- The all group of XSD:

```xml
<xsd:complexType name="PurchaseOrderType">
  <xsd:all>
    <xsd:element name="billTo" type="USAddress"/>
    <xsd:element ref="comment" minOccurs="0"/>
    <xsd:element name="items" type="Items"/>
  </xsd:all>
</xsd:complexType>
```
The cost of Interleaving

- Membership
  - RE : PTime
  - RE with & : NP-complete

- Inclusion
  - RE: PSPACE (EXPTIME for EDTDs) complete
  - RE with & : EXPSPACE complete

- Our conflict-free expressions:
  - Inclusion: quadratic [IS09]
  - Membership: linear [CIKM08]
Our conflict-free REs

\[
T ::= \varepsilon \mid a[m..n] \mid T + T \mid T \cdot T \mid T&T
\]
Our conflict-free REs

T ::= $\epsilon$ | a[m..n] | T + T | T·T | T&T

- Two restrictions:
  1. repetition T* restricted to a* (denoting $a[1..*] + \varepsilon$)
  2. single occurrence:
     (a+b·a+a·c) : no
     (a·b?) : ok

- Are these restrictions acceptable?

[BexNevenSchwentickTuyls-VLDB06]: “An examination of 819 DTDs and XSDs … more than 99% of the REs occurring in practical schema’s are CHAREs”
Types as constraints

\[ T = ((a[1..3] \cdot b[2..2]) + c[1..*]) \quad \text{and} \quad w \text{ in } L(T) \]
Types as constraints

\[ T = ((a[1..3] \cdot b[2..2]) + c[1..*]) \text{ and } w \text{ in } L(T) \]

- **lower-bound** (*nillability*): at least one of \( \{a, b, c\} = S(T) \) is in \( w \);
Types as constraints

\[ T = ((a[1..3] \cdot b[2..2]) + c[1..*]) \quad \text{and} \quad w \text{ in } L(T) \]

- lower-bound (nillability): at least one of \( \{a, b, c\} = S(T) \) is in \( w \);
- upper-bound: no symbol out of \( \{a, b, c\} \) is in \( w \);
Types as constraints

\[ T = ((a[1..3] \cdot b[2..2]) + c[1..*]) \quad \text{and} \quad w \text{ in } L(T) \]

- **lower-bound** (*nillability*): at least one of \{a, b, c\} = S(T) is in w;
- **upper-bound**: no symbol out of \{a, b, c\} is in w;
- **cardinality**: if a is in w, it appears 1, 2 or 3 times; if b is there, it appears twice…
Types as constraints

\[ T = ((a[1..3] \cdot b[2..2]) + c[1..*]) \text{ and } w \in L(T) \]

- **lower-bound** (*nillability*): at least one of \{a,b,c\}=S(T) is in \(w\);
- **upper-bound**: no symbol out of \{a, b, c\} is in \(w\);
- **cardinality**: if \(a\) is in \(w\), it appears 1, 2 or 3 times; if \(b\) is there, it appears twice…
- **exclusion**: any of \(\{a,b\}\) excludes \(c\)
  
  \(c\) excludes any of \(\{a,b\}\)
Types as constraints

\[ T = ((a[1..3] \cdot b[2..2]) + c[1..*]) \] and \( w \) in \( L(T) \)

- **lower-bound** (nillability): at least one of \( \{a, b, c\} = S(T) \) is in \( w \);
- **upper-bound**: no symbol out of \( \{a, b, c\} \) is in \( w \);
- **cardinality**: if \( a \) is in \( w \), it appears 1, 2 or 3 times; if \( b \) is there, it appears twice…
- **exclusion**: any of \( \{a, b\} \) excludes \( c \)
  \( c \) excludes any of \( \{a, b\} \)
- **co-occurrence**: \( a \) requires \( b \); \( b \) requires \( a \)
Types as constraints

\[ T = ((a[1..3]\cdot b[2..2]) + c[1..*]) \text{ and } w \text{ in } L(T) \]

- **lower-bound** (*nillability*): at least one of \{a, b, c\} = S(T) is in w;
- **upper-bound**: no symbol out of \{a, b, c\} is in w;
- **cardinality**: if a is in w, it appears 1, 2 or 3 times; if b is there, it appears twice…
- **exclusion**: any of \{a, b\} excludes c
  c excludes any of \{a, b\}
- **co-occurrence**: a requires b; b requires a
- **order**: any a comes before any b
Types as constraints

\[ T = ((a[1..3] \cdot b[2..2]) + c[1..*]) \text{ and } w \text{ in } L(T) \]

- **lower-bound** \((nillability)\): at least one of \(\{a, b, c\} = S(T)\) is in \(w\);
- **upper-bound**: no symbol out of \(\{a, b, c\}\) is in \(w\);
- **cardinality**: if \(a\) is in \(w\), it appears 1, 2 or 3 times; if \(b\) is there, it appears twice…
- **exclusion**: any of \(\{a, b\}\) excludes \(c\)
  - \(c\) excludes any of \(\{a, b\}\)
- **co-occurrence**: \(a\) requires \(b\); \(b\) requires \(a\)
- **order**: any \(a\) comes before any \(b\)

This is a complete characterization of \(T\)!
Types as constraints

\[ T = ((a[1..3] \cdot b[2..2]) + c[1..*]) \] and \( w \) in \( L(T) \)

- **lower-bound** \( S(T) \)
- **upper-bound**: \( \text{Upper}(S(T)) \)
- **cardinality**: \( a[1..3] \land b[2..2] \land c[1..*] \)
- **exclusion**: \( \{a,b\} \prec \succ \{c\} \)
- **co-occurrence**: \( a \Rightarrow b \land b \Rightarrow a \) (abbreviated as \( a \Leftrightarrow b \))
- **order**: \( a < b \)
Constraints

F ::= A | A⇒B | a?[m..n] | upper(A) | A<B | F ∧ F'
Constraints

F ::= A | A ⇒ B | a?[m..n] | upper(A) | A < B | F ∧ F'
Derived operators

- Double co-occurrence:
  - $A \iff B \iff_{\text{def}} A \Rightarrow B$ and $B \Rightarrow A$

- Mutual exclusion
  - $A \not<\not> B \iff_{\text{def}} A < B$ and $B < A$
  - corresponds to c-f union types $T_A + T_B$

- Negation
  - $\neg A \iff_{\text{def}} A \Rightarrow \emptyset$
  - False $\iff_{\text{def}} \emptyset$
  - True $\iff_{\text{def}} \emptyset \Rightarrow \emptyset$
Flat, order and co-occurrence constraints

- Each c-f type can be associated to a conjunction
  \[ F(T) = \text{Flat}(T) \land \text{OC}(T) \land \text{CC}(T) \]

- Theorem [IS09]: \( w \in T \iff w \models F(T) \)

- Theorem (subtyping)
  \[ T \subseteq U \iff T \models \text{Ffat}(U), T \models \text{OC}(T), T \models \text{CC}(T) \]

- Each of the 3 above entailments can be checked: independently and in \( O(n^2) \) time [IS09].

- In a recent work [ICDT09]: quadratic algorithm when only \( U \) is c-f

- Good news for result analysis: \( Q : S \rightarrow S' \) via \( S_{out} \subseteq S' \)
Constraints for efficient validation
Main points

- XML schema validation \(\simeq\) RE membership
- Membership for RE+\{interleaving, counting\} is NP-complete
- Most of REs defined in real-life schemas are conflict-free
- Semantics of c-f REs can be captured by logical constraints (previous work at DBPL’07)
- Streaming RE membership checking via streaming constraint residuation
- Linear complexity!
- Extension to XML schema validation: immediate (see [CIKM08])
Constraints construction

\[ T = ((a+\varepsilon)\& b[1..5]) + (c\cdot d[1..\ast]) \]
Constraints construction

\[ T = ((a+\varepsilon)b[1..5]) + (c\cdot d[1..*]) \]
Constraints construction

\[ T = ((a+\varepsilon)\&b[1..5]) + (c\cdot d[1..*]) \]
Constraints construction

\[ T = ((a + \varepsilon) \& b[1..5]) + (c \cdot d[1..*]) \]
T = ((a + ε) & b[1..5]) + (c · d[1..*])

T is not nilable: ε ∈ T

C(T) = abcd ∧ ......
Constraints construction

\[ T = ((a+\varepsilon)&b[1..5]) + (c\cdot d[1..\ast]) \]

\[ T \text{ is not nillable: } \varepsilon \notin T \]

\[ C(T) = abcd \land \text{upper}(abcd) \]

\[ \text{abcd } \land \text{upper}(abcd) \]
Constraints construction

\[ T = ((a+\varepsilon)\&b[1..5]) + (c\cdot d[1..\ast]) \]

\[ C(T) = abcd \land upper(abcd) \land a[1..1] \land \ldots \land d[1..\ast] \]
Constraints construction

\[ T = ((a+\varepsilon)\&b[1..5]) + (c\cdot d[1..\ast]) \]

\[ C(T) = \text{abcd} \land \text{upper(abcd)} \land a[1..1] \land \ldots \land d[1..\ast] \land a \Rightarrow b \land c \Leftrightarrow d \]
Constraints construction

\[ T = ((a+\varepsilon)\&b[1..5]) + (c\cdot d[1..\ast]) \]

\[ C(T) = abcd \land \text{upper}(abcd) \land a[1..1] \land \ldots \land d[1..\ast] \land ab \not\sim cd \land c < d \land a \Rightarrow b \land c \Leftrightarrow d \]

Theorem: \( w \in T \iff w \models C(T) \)
Constraint membership

- We consider $F = C(T)$ instead of $T$
Constraint membership

- We consider $F = C(T)$ instead of $T$
- We build a tree representation of $C(T)$
Constraint membership

- We consider $F = C(T)$ instead of $T$
- We build a tree representation of $C(T)$
- We check $w = a_1 \cdot a_2 \cdot \ldots \cdot a_n \models C(T)$ in a streaming fashion
Constraint membership

- We consider $F = C(T)$ instead of $T$
- We build a tree representation of $C(T)$
- We check $w = a_1 \cdot a_2 \cdot \ldots \cdot a_n \not\models C(T)$ in a streaming fashion

$F \xrightarrow{a_1} F_1$
Constraint membership

- We consider $F = C(T)$ instead of $T$
- We build a tree representation of $C(T)$
- We check $w = a_1 \cdot a_2 \cdot \ldots \cdot a_n \not\models C(T)$ in a streaming fashion

$F \xrightarrow{a_1} F1 \xrightarrow{a_2} F2$
Constraint membership

- We consider $F = C(T)$ instead of $T$
- We build a tree representation of $C(T)$
- We check $w = a_1 \cdot a_2 \cdot \ldots \cdot a_n \not\models C(T)$ in a streaming fashion

\[ F \xrightarrow{a_1} F_1 \xrightarrow{a_2} F_2 \xrightarrow{a_3} \ldots \xrightarrow{a_n} F_n \]
Constraint membership

- We consider $F = C(T)$ instead of $T$
- We build a tree representation of $C(T)$
- We check $w = a_1 \cdot a_2 \cdot \ldots \cdot a_n \models C(T)$ in a streaming fashion

$F \xrightarrow{a_1} F_1 \xrightarrow{a_2} F_2 \xrightarrow{a_3} \ldots \xrightarrow{a_n} F_n$
We consider $F = C(T)$ instead of $T$

We build a tree representation of $C(T)$

We check $w = a_1 \cdot a_2 \cdot \ldots \cdot a_n \models C(T)$ in a streaming fashion

Constraint membership
We consider $F = C(T)$ instead of $T$

- We build a tree representation of $C(T)$
- We check $w = a_1 \cdot a_2 \cdot \ldots \cdot a_n \not\models C(T)$ in a streaming fashion

**Constraint membership**

\[
\begin{align*}
F & \xrightarrow{a_1} F_1 & \xrightarrow{a_2} F_2 & \ldots & \xrightarrow{a_n} F_n \\
residuals & & & &
\end{align*}
\]

\[
w \not\models C(T) \iff F_n \cap \{\text{False, A}\} = \emptyset
\]
Constraint membership

- We consider $F = C(T)$ instead of $T$
- We build a tree representation of $C(T)$
- We check $w = a_1 \cdot a_2 \cdot \ldots \cdot a_n \models C(T)$ in a streaming fashion

- Important: flat constraints do not need residuation:
  - counting constraints $a?([m..n])$: keep some counters updated during the visit
  - lower and upper bound constraints: trivial.
# Residuation

<table>
<thead>
<tr>
<th>Input</th>
<th>Constraint</th>
<th>Residual after $a_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a_i \in A$</td>
<td>$A \Rightarrow B$</td>
<td>$B$</td>
</tr>
<tr>
<td>$a_i \in B$</td>
<td>$A \Rightarrow B$</td>
<td>true</td>
</tr>
<tr>
<td>$a_i \in A$</td>
<td>$A \Leftrightarrow B$</td>
<td>$B$</td>
</tr>
<tr>
<td>$a_i \in A$</td>
<td>$A$</td>
<td>true</td>
</tr>
<tr>
<td>$a_i \in A$</td>
<td>$A \nLeftarrow B$</td>
<td>$\neg B$</td>
</tr>
<tr>
<td>$a_i \in B$</td>
<td>$A &lt; B$</td>
<td>$\neg A$</td>
</tr>
<tr>
<td>$a_i \in A$</td>
<td>$\neg B$</td>
<td>$\neg B$</td>
</tr>
<tr>
<td>$a_i \in A$</td>
<td>$\neg A$</td>
<td>false</td>
</tr>
</tbody>
</table>
# Residuation

<table>
<thead>
<tr>
<th>Input</th>
<th>Constraint</th>
<th>Residual after (a_i)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a_i\in A)</td>
<td>(A\implies B)</td>
<td>(B)</td>
</tr>
<tr>
<td>(a_i\in B)</td>
<td>(A\implies B)</td>
<td>true</td>
</tr>
<tr>
<td>(a_i\in A)</td>
<td>(A\iff B)</td>
<td>(B)</td>
</tr>
<tr>
<td>(a_i\in A)</td>
<td>(A)</td>
<td>true</td>
</tr>
<tr>
<td>(a_i\in A)</td>
<td>(A &lt;&gt; B)</td>
<td>(\neg B)</td>
</tr>
<tr>
<td>(a_i\in B)</td>
<td>(A &lt; B)</td>
<td>(\neg A)</td>
</tr>
<tr>
<td>(a_i\in A)</td>
<td>(\neg B)</td>
<td>(\neg B)</td>
</tr>
<tr>
<td>(a_i\in A)</td>
<td>(\neg A)</td>
<td>false</td>
</tr>
</tbody>
</table>
## Residuation

<table>
<thead>
<tr>
<th>Input</th>
<th>Constraint</th>
<th>Residual after $a_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a_i \in A$</td>
<td>$A \implies B$</td>
<td>$B$</td>
</tr>
<tr>
<td>$a_i \in B$</td>
<td>$A \implies B$</td>
<td>true</td>
</tr>
<tr>
<td>$a_i \in A$</td>
<td>$A \iff B$</td>
<td>$B$</td>
</tr>
<tr>
<td>$a_i \in A$</td>
<td>$A$</td>
<td>true</td>
</tr>
<tr>
<td>$a_i \in A$</td>
<td>$A \not&lt; B$</td>
<td>$\neg B$</td>
</tr>
<tr>
<td>$a_i \in B$</td>
<td>$A \not&lt; B$</td>
<td>$\neg A$</td>
</tr>
<tr>
<td>$a_i \in A$</td>
<td>$\neg B$</td>
<td>$\neg B$</td>
</tr>
<tr>
<td>$a_i \in A$</td>
<td>$\neg A$</td>
<td>false</td>
</tr>
</tbody>
</table>
Residuation

<table>
<thead>
<tr>
<th>Input</th>
<th>Constraint</th>
<th>Residual after $a_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a_i \in A$</td>
<td>$A \Rightarrow B$</td>
<td>$B$</td>
</tr>
<tr>
<td>$a_i \in B$</td>
<td>$A \Rightarrow B$</td>
<td>true</td>
</tr>
<tr>
<td>$a_i \in A$</td>
<td>$A \Leftrightarrow B$</td>
<td>$B$</td>
</tr>
<tr>
<td>$a_i \in A$</td>
<td>$A$</td>
<td>true</td>
</tr>
<tr>
<td>$a_i \in A$</td>
<td>$A &lt; &gt; B$</td>
<td>$\neg B$</td>
</tr>
<tr>
<td>$a_i \in B$</td>
<td>$A &lt; B$</td>
<td>$\neg A$</td>
</tr>
<tr>
<td>$a_i \in A$</td>
<td>$\neg B$</td>
<td>$\neg B$</td>
</tr>
<tr>
<td>$a_i \in A$</td>
<td>$\neg A$</td>
<td>false</td>
</tr>
</tbody>
</table>
## Residuation

<table>
<thead>
<tr>
<th>Input</th>
<th>Constraint</th>
<th>Residual after $a_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a_i \in A$</td>
<td>$A \Rightarrow B$</td>
<td>B</td>
</tr>
<tr>
<td>$a_i \in B$</td>
<td>$A \Rightarrow B$</td>
<td>true</td>
</tr>
<tr>
<td>$a_i \in A$</td>
<td>$A \Leftrightarrow B$</td>
<td>B</td>
</tr>
<tr>
<td>$a_i \in A$</td>
<td>$A$</td>
<td>true</td>
</tr>
<tr>
<td>$a_i \in A$</td>
<td>$A \leftrightarrow B$</td>
<td>$\neg B$</td>
</tr>
<tr>
<td>$a_i \in B$</td>
<td>$A &lt; B$</td>
<td>$\neg A$</td>
</tr>
<tr>
<td>$a_i \in A$</td>
<td>$\neg B$</td>
<td>$\neg B$</td>
</tr>
<tr>
<td>$a_i \in A$</td>
<td>$\neg A$</td>
<td>false</td>
</tr>
</tbody>
</table>
# Residuation

<table>
<thead>
<tr>
<th>Input</th>
<th>Constraint</th>
<th>Residual after $a_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a_i \in A$</td>
<td>$A \Rightarrow B$</td>
<td>$B$</td>
</tr>
<tr>
<td>$a_i \in B$</td>
<td>$A \Rightarrow B$</td>
<td>true</td>
</tr>
<tr>
<td>$a_i \in A$</td>
<td>$A \Leftrightarrow B$</td>
<td>$B$</td>
</tr>
<tr>
<td>$a_i \in A$</td>
<td>$A$</td>
<td>true</td>
</tr>
<tr>
<td>$a_i \in A$</td>
<td>$A \nRightarrow B$</td>
<td>$\neg B$</td>
</tr>
<tr>
<td>$a_i \in B$</td>
<td>$A \nRightarrow B$</td>
<td>$\neg A$</td>
</tr>
<tr>
<td>$a_i \in A$</td>
<td>$\neg B$</td>
<td>$\neg B$</td>
</tr>
<tr>
<td>$a_i \in A$</td>
<td>$\neg A$</td>
<td>false</td>
</tr>
</tbody>
</table>
### Residuation

<table>
<thead>
<tr>
<th>Input</th>
<th>Constraint</th>
<th>Residual after $a_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a_i \in A$</td>
<td>$A \Rightarrow B$</td>
<td>$B$</td>
</tr>
<tr>
<td>$a_i \in B$</td>
<td>$A \Rightarrow B$</td>
<td>true</td>
</tr>
<tr>
<td>$a_i \in A$</td>
<td>$A \Leftrightarrow B$</td>
<td>$B$</td>
</tr>
<tr>
<td>$a_i \in A$</td>
<td>$A$</td>
<td>true</td>
</tr>
<tr>
<td>$a_i \in A$</td>
<td>$A \not&lt; \not&gt; B$</td>
<td>$\neg B$</td>
</tr>
<tr>
<td>$a_i \in B$</td>
<td>$A &lt; B$</td>
<td>$\neg A$</td>
</tr>
<tr>
<td>$a_i \in A$</td>
<td>$\neg B$</td>
<td>$\neg B$</td>
</tr>
<tr>
<td>$a_i \in A$</td>
<td>$\neg A$</td>
<td>false</td>
</tr>
</tbody>
</table>
Residuation

<table>
<thead>
<tr>
<th>Input</th>
<th>Constraint</th>
<th>Residual after $a_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a_i \in A$</td>
<td>$A \Rightarrow B$</td>
<td>$B$</td>
</tr>
<tr>
<td>$a_i \in B$</td>
<td>$A \Rightarrow B$</td>
<td>$true$</td>
</tr>
<tr>
<td>$a_i \in A$</td>
<td>$A \Leftrightarrow B$</td>
<td>$B$</td>
</tr>
<tr>
<td>$a_i \in A$</td>
<td>$A$</td>
<td>$true$</td>
</tr>
<tr>
<td>$a_i \in A$</td>
<td>$A \not&lt; B$</td>
<td>$\neg B$</td>
</tr>
<tr>
<td>$a_i \in B$</td>
<td>$A &lt; B$</td>
<td>$\neg A$</td>
</tr>
<tr>
<td>$a_i \in A$</td>
<td>$\neg B$</td>
<td>$\neg B$</td>
</tr>
<tr>
<td>$a_i \in A$</td>
<td>$\neg A$</td>
<td>$false$</td>
</tr>
</tbody>
</table>
## Residuation

<table>
<thead>
<tr>
<th>Input</th>
<th>Constraint</th>
<th>Residual after $a_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a_i \in A$</td>
<td>$A \Rightarrow B$</td>
<td>$B$</td>
</tr>
<tr>
<td>$a_i \in B$</td>
<td>$A \Rightarrow B$</td>
<td>true</td>
</tr>
<tr>
<td>$a_i \in A$</td>
<td>$A \Leftrightarrow B$</td>
<td>$B$</td>
</tr>
<tr>
<td>$a_i \in A$</td>
<td>$A$</td>
<td>true</td>
</tr>
<tr>
<td>$a_i \in A$</td>
<td>$A &lt; &gt; B$</td>
<td>$\neg B$</td>
</tr>
<tr>
<td>$a_i \in B$</td>
<td>$A &lt; B$</td>
<td>$\neg A$</td>
</tr>
<tr>
<td>$a_i \in A$</td>
<td>$\neg B$</td>
<td>$\neg B$</td>
</tr>
<tr>
<td>$a_i \in A$</td>
<td>$\neg A$</td>
<td>false</td>
</tr>
</tbody>
</table>
Linear residuation

\[ d \in ((a+\varepsilon) \& b[1..5]) + (c \cdot d[1..*]) \]

Diagram: (see image for the graphical representation of the expression)
Linear residuation

d ∈ ((a+ε)&b[1..5]) + (c·d[1..*])
Linear residuation

d \in ((a+\varepsilon)\& b[1..5]) + (c \cdot d[1..*])

\begin{align*}
& a \in B \\
& A \Leftrightarrow B \\
& A
\end{align*}
Linear residuation

\[ d \in ((a+\epsilon) \& b[1..5]) + (c \cdot d[1..*]) \ ? \]
Linear residuation

\[ d \in ((a+\epsilon)&b[1..5]) + (c\cdot d[1..*]) \ ? \]
Linear residuation

d \in ((a+\varepsilon)\&b[1..5]) + (c\cdot d[1..\ast]) ?

\[ a \in A \quad A \leftrightarrow B \quad \neg B \]
Linear residuation

d ∈ ((a+ε)&b[1..5]) + (c·d[1..*])?
Linear residuation

\[ d \not\in ((a + \varepsilon) \& b[1..5]) + (c \cdot d[1..\ast]) \]

Failure : the final residual contains a formula A=c
Linear residuation

\[ abb \in ((a+\varepsilon) \& b[1..5]) + (c \cdot d[1..\ast]) \]
Linear residuation

\[abb \in ((a+\varepsilon)\&b[1..5]) + (c\cdot d[1..*]) \ ?\]
Linear residuation

\[ a b b \in ((a + \varepsilon) \& b[1..5]) + (c \cdot d[1..*]) \ ? \]
\[ abb \in ((a+\varepsilon) \& b[1..5]) + (c \cdot d[1..\ast]) \]
Linear residuation

\[ abb \in ((a + \varepsilon) \& b[1..5]) + (c \cdot d[1..*]) \]
Linear residuation

\[
abb \in ((a+\varepsilon)\&b[1..5]) + (c\cdot d[1..*])
\]
Linear residuation

\[ abb \in ((a+\varepsilon)&b[1..5]) + (c\cdot d[1..*]) \ ? \]
Linear residuation

\[ abb \in ((a + \varepsilon) \& b[1..5]) + (c \cdot d[1..*]) \]
Linear residuation

\[ a b b \in ((a + \varepsilon) \& b[1..5]) + (c \cdot d[1..\ast]) \]
Linear residuation

\[ a \cdot b \in ((a + \varepsilon) \& b[1..5]) + (c \cdot d[1..*]) \]
Linear residuation

\[ a \in ((a + \varepsilon) \& b^{1..5}) + (c \cdot d^{1..*}) \]
Linear residuation

\( a b b \in ((a+\varepsilon)\&b[1..5]) + (c\cdot d[1..*]) \quad ? \)

\[
(a_i \in A \quad A \quad true)
\]
Linear residuation

\[ a^b b \in ((a + \varepsilon) b[1..5]) + (c \cdot d[1..*]) \ ? \]
Linear residuation

\[ abb \in ((a+\varepsilon)&b[1..5]) + (c\cdot d[1..*]) \ ? \]

Nothing new!
Linear residuation

\[ abb \in ((a+\varepsilon)\& b[1..5]) \ominus (c \cdot d[1..\star]) \ ? \]
Linear residuation

\[ abb \in ((a+\varepsilon) \& b[1..5]) + (c \cdot d[1..*]) \quad ? \]
Linear residuation

\[ \text{Nat} \in ((a + \varepsilon) \& b[1..5]) + (c \cdot d[1..*]) \]

No A or False in the final residual ⇒ Success!
Linear residuation

$\text{abb} \in ((a+\varepsilon) \& b[1..5]) + (c \cdot d[1..*])$

Complexity: $O(|T|+|w| \cdot \text{depth}(T))$
Linear residuation

\[
abb \in ((a+\varepsilon)\&b[1..5]) + (c\cdot d[1..*]) \ ?
\]

Complexity: \( O(|T|+|w|\cdot \text{depth}(T)) \)

We can do better: \( O(|T|+|w|) \)!
Avoiding redundant visits

\[ abb \in ((a + \epsilon) \& b[1..5]) + (c \cdot d[1..*]) \]

Remark: the operations made for the second b were redundant!
Avoiding redundant visits

\[ abb \in ((a+\varepsilon)\&b[1..5]) + (c\cdot d[1..\star]) \]

Remark: the operations made for the second b were redundant!

Once a symbol is met and processed, there is almost no reason to consider and process it if met again.
Stability

- A node $n$ is visited each time a symbol in its left or right hand side sub-tree is met in $w$
Stability

- A node $n$ is visited each time a symbol in its left or right hand side sub-tree is met in $w$
Stability

\[ a \in A \]
Stability

This is almost always true!

It is almost always true that $n$ does not need to be visited more than once for symbols in $A$
Stability

\[ a, a' \in A \]

Redundant transition!
Stability - exception

\[ a, a' \in A \quad w \cap B \neq \emptyset \]

Residuation stops!

Exception: \( F = A < B \) and \( w \cap B \neq \emptyset \)
The linear algorithm

• B-stability always holds
• So during residuation each node needs to be processed/visited at most three times.
  • For the pattern A-B-A with F= A < B
• Complexity $O(|T|+|w|)$
Some tests without residuation

Figure 9: Scalability of Xelf.
Conclusions

- We have seen some main ideas behind the use of XML schema:
  - checking query correctness and result analysis
  - efficient document validation
  - query and update optimisation (time/space)

- Other interesting applications:
  - schema mapping maintenance in XML data integration (based on result analysis and a notion of type-projection) [DBPL05, PPDP06, TOIT09]
  - query-update independence: a technique derived from type-based projection can ensure highly precise analysis [work in progress, ask Federico Ulliana for details]
References


References


Merci ! Any questions ?